Enhancing Particle Swarm Optimization for Portfolio Optimization (Deng, Lin, Lo, 2012)

Alessandro Rossato, n. 875067

May 28, 2024

Summary

- **Paper Presentation**
  - Markowitz Portfolio Problem
  - PSO variants
  - Experiments and Results
- **Re-Implementation**
  - Proposed PSO vs Implemented PSO
  - Comparison with the Efficient Frontier
  - Critique and Limitations

Introduction

- **Algorithm:** Particle Swarm Optimization (PSO) is computational method inspired by the social behavior of birds. Each particle in the swarm represents a potential solution, flying through a multidimensional search space.
- **Application:** PSO is widely used in portfolio optimization due to its simplicity and effectiveness.
- **Objective:** This paper proposes several enhanced PSO variants to improve performance in Cardinality Constraints Markowitz Portfolio Optimization Problem and compare them with other algorithms like GA, SA and TS.

Overview on Markowitz Portfolio Problem

- **Objective:** Under some constraints, minimize risk for a given level of return or maximize the return for a given level of risk (levels are defined by the λ parameter).
- **Return:** Expected portfolio return.

$$R_p = \sum_{i=1}^{n} x_i R_i$$

- **Risk:** Variance of portfolio return.

$$\sigma_p^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} x_i x_j \sigma_{ij}$$

- **Risk Aversion Parameter (λ):** Represents the investor's aversion to risk.

Constraints in MPO

- **Budget Constraint:** Total investment should not exceed available funds.

$$\sum_{i=1}^{n} x_i = 1$$

- **Cardinality Constraint:** Limit the number of assets k in the portfolio.

$$\sum_{i=1}^{n} z_i = k$$

- **Holdings Constraint:** Limit the range of admissible proportion of assets (avoid short selling and corner portfolio)

$$\epsilon_i z_i \leq x_i \leq \delta_i z_i \quad i = 1, \ldots, N$$

where x_i is the weight of the asset, z_i is a binary variable indicating if asset i is included in the portfolio and the parameters ϵ_i and δ_i represents minimum and maximum exposure.

CCMPO Formulation

$$\textbf{Minimize} \quad \lambda \left[\sum_{i=1}^{N} \sum_{j=1}^{N} x_i x_j \sigma_{ij} \right] - (1-\lambda) \left[\sum_{i=1}^{N} x_i u_i \right]$$

subject to

$$\textbf{Budget} \quad \sum_{i=1}^{N} x_i = 1$$

$$\textbf{Cardinality} \quad \sum_{i=1}^{N} z_i = K$$

$$z_i \in \{0,1\} \quad i = 1, \ldots, N$$

$$\textbf{Holdings} \quad \epsilon_i z_i \le x_i \le \delta_i z_i \quad i = 1, \ldots, N$$

$$0 \le \epsilon_i \le \delta_i \le 1 \quad i = 1, \ldots, N$$

Introduction to PSO Variants

- The authors explores four PSO variants tailored for portfolio optimization:
  1. Basic PSO (Constant Inertia and Maximum Velocity)
  2. PSO-DIV (Dynamic Inertia and Velocity Reduction)
  3. PSO-C (Constriction Coefficient)
  4. Proposed PSO with Reflection and Mutation Strategies
- Each variant addresses specific challenges in portfolio optimization, aiming to **improve optimization** performance and **overcome limitations**.
- All the models start with particle that have **random** positions within bound and **zero** velocity and terminates when **no improvement** occurs over repeated iterations

Basic PSO

- At each iteration t, the **position** $x_{t+1_{i,j}}$ of the ith particle in dimension j is updated using its velocity $v_{t+1_{i,j}}$:

$$x_{t+1_{i,j}} = x_{t_{i,j}} + v_{t+1_{i,j}}$$

- The **velocity update** rule depends on:
  - Current velocity $v_{t_{i,j}}$
  - Differences between personal best $p_{i,j}$ and global best pg_j positions respect to the current position
  - Positive acceleration coefficients c_1 and c_2 (trust parameters)
  - Random values r_1 and r_2 in range [0, 1]
- To control exploitation and exploration, an **inertia weight** w and **maximum velocity** limit $v_{\max}$ are introduced:

$$v_{t+1_{i,j}} = w \cdot v_{t_{i,j}} + c_1 \cdot r_1 \cdot (p_{i,j} - x_{t_{i,j}}) + c_2 \cdot r_2 \cdot (pg_j - x_{t_{i,j}})$$

$$v_{\max_j} = d \cdot (x_{\max_j} - x_{\min_j})$$

PSO-DIV (Dynamic Inertia and Velocity Reduction)

- Fourie and Groenwold (2002) suggested a **dynamic inertia** weight and maximum **velocity reduction** if no improvement occurs in global best solution, as follows:

  $$\text{If } f(p_t) = f(p_{t-h}), \text{ then } x_{t+1} = \alpha \cdot w_t \text{ and } v_{\max,j} = \beta \cdot v_{\max,j}$$

  where α and β are such that $0 < \alpha, \beta < 1$.

PSO-C (Constriction Coefficient PSO)

- Clerc and Kennedy (2002) modified the basic PSO by introducing a **constriction coefficient** χ.
- The **velocity update** equation changes to:

$$v_{t+1_{ij}} = \chi \left[v_{t_{ij}} + \phi_1 (p_{ij} - x_{t_{ij}}) + \phi_2 (p_{gj} - x_{t_{i,j}}) \right]$$

where:

$$\chi = \frac{2k}{|2 - \phi - \sqrt{\phi^2 - 4\phi}|}, \quad \phi = \phi_1 + \phi_2, \quad \phi_1 = c_1 r_1, \quad \phi_2 = c_2 r_2$$

- Conditions $\Phi \geq 4$ and $k \in [0, 1]$ ensure swarm convergence.
- Parameter k controls swarm behaviour: for $k \approx 0$, fast convergence is achieved with local exploitation; conversely, $k \approx 1$ leads to slow convergence with high exploration.

Proposed PSO

- **Problem with CCMPO:** PSO quickly stagnates to the local optimum in order to satisfy the Cardinality constraint in the portfolio, especially when are consider high values for risk aversion parameter λ.
- **Four extension** are proposed:
  - New strategies for constraints satisfaction
  - Time-Variant Inertia weight (w)
  - Time-Variant Acceleration Coefficients (c1 and c2)
  - Mutation operator
- Explain in details later in the re-implementation phase

Performance Metrics

- **Accuracy:** Measures the closeness of the obtained solutions to the optimal solution for return and risk.

$$\text{Accuracy} = \frac{\text{Obtained Solution Value}}{\text{Optimal Solution Value}} \times 100\%$$

- **Robustness:** Evaluates the consistency of the algorithm across multiple runs.

$$\text{Robustness} = \text{Standard Deviation of Solution Values}$$

- **Diversity:** Assesses the variety of solutions generated by the algorithm.

$$\text{diversity}(S) = \frac{1}{n_s} \sum_{i=1}^{n_s} \sqrt{\sum_{j=1}^{n_x} (x_{i,j} - \bar{x}_j)^2}$$

Where $\bar{x}_j = \frac{1}{n_s} \sum_{i=1}^{n_s} x_{i,j}$.

Computational Experiments

- **Experiment Definition:**
  - The PSO searched for efficient frontiers by testing 50 different values of the risk aversion parameter k in the cardinality-constrained Markowitz portfolio model.
  - Employed five benchmark datasets from (Chang et al., 2000; Fernandez Gomez, 2007).
  - Data correspond to weekly price data (March 1992 - September 1997) for the following indices:
    - Hang Seng 31 (Hong Kong)
    - DAX 100 (Germany)
    - FTSE 100 (UK)
    - SP 100 (USA)
    - Nikkei 225 (Japan)
  - Number of assets (N): 31, 85, 89, 98, and 225, respectively.
- **Common Parameters:**
  - $K = 10$
  - $\epsilon_i = 0.01$
  - $\delta_i = 1$

Experiment 1: Comparison of PSO Variants

- **Parameters:**
  - **Basic PSO:** $(w, c_1, c_2, v_{max}) = (0.7298, 1.49618, 1.49618, 1)$
  - **PSO-DIV:** $a = b = 0.99, h = 10,$ initial $v_{max} = 1$
  - **PSO-Constriction:** $c_1 = 2.8, c_2 = 1.3$
  - **Proposed PSO:** $c_{1,\text{max}} = c_{2,\text{max}} = 2.5, c_{1,\text{min}} = c_{2,\text{min}} = 0.5, w_{initial} = 0.9$ linearly decreased to $w_{final} = 0.4, b = 5$
- **Swarm Size and Termination:**
  - Swarm size: 100 for all PSOs
  - Termination: No improvement over 100 iterations
- **Results:**
  - Proposed PSO generally achieved lower minimum mean percentage error compared to variant PSOs.
  - Average CPU times and number of iterations of convergence were comparable among all methods.
- **Effect of Risk Aversion Parameter λ:**
  - Low λ: Portfolio emphasizes maximizing return regardless of risk, resulting in fewer significant investments (exceeded $1/k$).
  - High λ: Portfolio emphasizes minimizing risk, resulting in diversity close to k assets.

Experiment 2: Comparative Performance with Other Heuristics

To compare the proposed PSO with other heuristics, the same data sets were considered in the constrained portfolio problem.

- **Heuristics Compared:**
  - Genetic Algorithm (GA)
  - Simulated Annealing (SA)
  - Tabu Search (TS)
- **Results:**
  - The proposed PSO almost always obtained the best performance in most cases.
- **Performance Metrics:**
  - The results on GA, SA, and TS are from Chang et al. (2000).
  - The proposed PSO was run for 1000 iterations and the results were averaged over 25 trials.

Re-Implementation - Introduction

- Basic and the proposed PSO are implemented:
  - All the constraints and strategies are replicated
  - Some parameters are slightly different in proposed PSO
  - Termination criteria and fitness function are freely adapted because no good explanation are provided in the paper
  - All the code is elaborated from scratch
- Data: S&P 500 from 2019-02-01 to 2024-02-01 (533 assets: includes all the firms that enter in the index)

Differences Proposed vs Implemented

Parameter	Proposed PSO	Implemented PSO
`num_iterations`	1000	5000
`num_assets`	`From 31 to 225`	532
`num_selected_assets`	10	50
`risk_aversion_values`	50	30

Handling Boundary Constraints - 1

- **Reflection Strategy** (Paterlini and Krink, 2006):
  - Applied during the initial search phase to explore a larger search area and to escape from local minima.
  - If the new position leaves the search space domain, reflect it back:

$$x_{i,j}^t = \begin{cases} x_{i,j}^t + 2(x_{l,j} - x_{i,j}^t) & \text{if } x_{i,j}^t < x_{l,j} \\ x_{i,j}^t - 2(x_{i,j}^t - x_{u,j}) & \text{if } x_{i,j}^t > x_{u,j} \end{cases}$$

    where $x_{u,j}$ and $x_{l,j}$ are the upper and lower bounds of each j-th component.
- **Final Boundary Adjustment:**
  - Reflection strategy terminates when no improvement is observed after numerous iterations and the values settled are:

$$x_{i,j}^t = \begin{cases} x_{l,j} & \text{if } x_{i,j}^t < x_{l,j} \\ x_{u,j} & \text{if } x_{i,j}^t > x_{u,j} \end{cases}$$

Handling Boundary Constraints - 2

```
for i = 1:num_particles
    % Update velocities
    particle_velocities(i, :) = w * particle_velocities(i, :) + ...
        c1 * rand * (personal_best_positions(i, :) - particle_positions(i, :)) + ...
        c2 * rand * (global_best_position - particle_positions(i, :));

    % Update positions
    if iter < num_iterations/3
        % Apply reflection strategy to prevent leaving search space domain
        particle_positions(i, :) = particle_positions(i, :) + particle_velocities(i, :);
        lower_out_of_bounds = particle_positions(i, :) < lower_bounds;
        upper_out_of_bounds = particle_positions(i, :) > upper_bounds;
        particle_positions(i, lower_out_of_bounds) = particle_positions(i, lower_out_of_bounds) + ...
            2 * (lower_bounds(lower_out_of_bounds) - particle_positions(i, lower_out_of_bounds));
        particle_positions(i, upper_out_of_bounds) = particle_positions(i, upper_out_of_bounds) - ...
            2 * (particle_positions(i, upper_out_of_bounds) - upper_bounds(upper_out_of_bounds));
    else
        % Apply Final Boundary Adjustment
        particle_positions(i, :) = particle_positions(i, :) + particle_velocities(i, :);
        particle_positions(i, :) = min(max(particle_positions(i, :), lower_bounds), upper_bounds);
    end
```

Figure: Positions and Velocities Updates

Handling Cardinality Constraints - 1

- Let Q be the set of K assets.
- Let K_{new} represent the number of assets after updating positions in the portfolio.
- **Adding Assets:**
  - $K_{\text{new}} < K$: Add assets to Q.
  - Randomly add asset $i \notin Q$.
  - Assign the minimum proportional value e_i to the new asset.
- **Removing Assets:**
  - $K_{\text{new}} > K$: Remove the smallest assets from Q until $K_{\text{new}} = K$.
- **Proportional Value Adjustment:**
  - $0 \leq \epsilon_i \leq x_i \leq \delta_i \leq 1$ for $i \in Q$.
  - Let s_i represent the proportion of the new position belonging to Q.
  - If $s_i < \epsilon_i$, replace asset s_i with ϵ_i.
  - If $s_i > \epsilon_i$, calculate the proportional share of the free portfolio:

$$x_i = \epsilon_i + \frac{s_i}{\sum_{j \in Q, s_i > \epsilon_i} s_i} \left(1 - \sum_{j \in Q}^{\epsilon_i} \right)$$

Handling Cardinality Constraints - 2

```
% Update positions within the cardinality constraint
current_set = find(binary_positions(i, :));
Knew = length(current_set);

% Adding Assets: if Knew < num_selected_assets
while Knew < num_selected_assets
    available_indices = setdiff(1:num_assets, current_set);
    new_asset = available_indices(randi(length(available_indices)));
    binary_positions(i, new_asset) = 1;
    particle_positions(i, new_asset) = epsilon; % Assign minimum proportional value
    current_set = find(binary_positions(i, :));
    Knew = length(current_set);
end

% Removing Assets: if Knew > num_selected_assets
while Knew > num_selected_assets
    [~, sorted_indices] = sort(particle_positions(i, current_set));
    asset_to_remove = current_set(sorted_indices(1));
    binary_positions(i, asset_to_remove) = 0;
    particle_positions(i, asset_to_remove) = 0;
    current_set = find(binary_positions(i, :));
    Knew = length(current_set);
end

% Proportional Value Adjustment
for j = current_set
    if particle_positions(i, j) < epsilon
        particle_positions(i, j) = epsilon;
    elseif particle_positions(i, j) > delta
        particle_positions(i, j) = delta;
    end
end

total_weight = sum(particle_positions(i, current_set));
if total_weight > 0
    particle_positions(i, current_set) = particle_positions(i, current_set) / total_weight;
end
```

Figure: Cardinality and weights Adjustment

Inertia Weight (w)

- The inertia weight w controls how previous velocity affects present velocity.
- **Time-variant w** (Shi and Eberhart, 2007):
  - w is linearly reduced during the search process.
  - Initially large w values decrease over time.
  - Encourages exploration initially and exploitation as time progresses.
- **Update formula** for w at time step t:

$$w(t) = (w(0) - w(n_t))\,\frac{n_t - t}{n_t} + w(n_t)$$

where:

- n_t is the maximum number of time steps.
- $w(0)$ is the initial inertia weight (usually 0.9).
- $w(n_t)$ is the final inertia weight (usually 0.4).

Acceleration Coefficients (c_1 and c_2)

- **Balancing Local and Global Search:**
  - If $c_1 > c_2$: Particles have a stronger attraction to their own best position, leading to excessive wandering.
  - If $c_2 > c_1$: Particles are more attracted to the global best position, causing premature convergence.
- **Time-Variant Strategy** (Ratnaweera et al., 2004):
  - Initially, c_1 is high and c_2 is low to focus on exploration.
  - Over time, c_1 decreases and c_2 increases to focus on exploitation.
- **Update formulas** for c_1 and c_2 at time step t:

$$c_1(t) = (c_{1,\min} - c_{1,\max})\frac{t}{n_t} + c_{1,\max}$$

$$c_2(t) = (c_{2,\max} - c_{2,\min})\frac{t}{n_t} + c_{2,\min}$$

where:

- n_t is the maximum number of time steps.
- $c_{1,\max} = c_{2,\max} = 2.5$.
- $c_{1,\min} = c_{2,\min} = 0.5$.

Inertia Weight and Acceleration Coefficients

```
for iter = 1:num_iterations
    fprintf('Starting iteration %d for lambda = %.2f\n', iter, risk_aversion_values(lambda));

    % Time-variant inertia weight
    w = (w_min - w_max) * (num_iterations - iter) / num_iterations + w_max;

    % Time-variant acceleration coefficients
    c1 = c1_max - (c1_max - c1_min) * (iter / num_iterations);
    c2 = c2_min + (c2_max - c2_min) * (iter / num_iterations);
```

Figure: Inertia and Acceleration Updates

Mutation Operator - 1

- **Mutation Process** (Tripathi et al., 2007):
  - Mutation operator is used to increase diversity.
  - For a given particle, a randomly chosen variable g_k is mutated as follows:

$$g'_k = \begin{cases} g_k + \Delta(t, \mathrm{UB} - g_k) & \text{if flip} = 0, \\ g_k + \Delta(t, g_k - \mathrm{LB}) & \text{if flip} = 1 \end{cases}$$

  - flip is a random binary event (0 or 1).
  - UB and LB are the upper and lower bounds of the variable g_k, respectively.
- **Function $\Delta(t, x)$:**

$$\Delta(t, x) = x\left(1 - r^{(1 - t/\mathrm{max\_t})^b}\right)$$

  - r is a random number in the range [0, 1].
  - max_t is the maximum number of iterations.
  - t is the current iteration number.
  - b determines the dependence of the mutation on the iteration number.

Mutation Operator - 2

```
% Mutation operator
if rand < mutation_probability
    gk_index = randi([1, num_assets]); % Randomly choose a variable index
    flip = randi([0, 1]); % Random event returning 0 or 1

    if flip == 0
        mutation_amount = delta_fun(iter, upper_bounds(gk_index) - ...
            particle_positions(i, gk_index), num_iterations, b);
        mutated_gk = particle_positions(i, gk_index) + mutation_amount;
    else
        mutation_amount = delta_fun(iter, particle_positions(i, gk_index) - ...
            lower_bounds(gk_index), num_iterations, b);
        mutated_gk = particle_positions(i, gk_index) + mutation_amount;
    end

    % Apply the mutation
    particle_positions(i, gk_index) = mutated_gk;
end
```

Termination and Re-Initialization

If no big improvement for 500 consecutive iteration, two way to proceed:

- **Re-Initialization criteria:** the solutions contain no viable weights
  - If poor fitness value stagnation
  - If the limit of 5 re-initialization for each level of risk aversion isn't reach yet
- **Breaking criteria:** the solutions exploit the space and reach a good result so can be terminated and proceeded with the next level of risk aversion
- Otherwise, all the 5000 iteration are performed for each levels

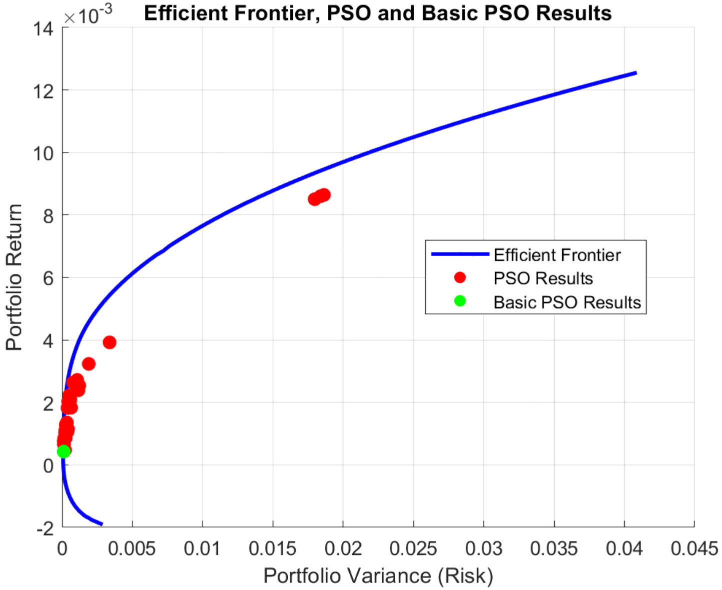
Fitness function

```
% Fitness function
    fitness = risk_aversion * variance/mean_variance - (1 - risk_aversion) * returns/mean_returns_val;

    % Check if sum of weights exceeds 1
    if sum(weights) > 1
        fitness = fitness + 10000 * sum(weights);
    end
    % Check if the number of selected assets exceeds the cardinality constraint
    if nnz(weights) > num_selected_assets
        fitness = fitness + 5000 * abs(nnz(weights)-num_selected_assets);

    end
    % Check if any weight is outside the bounds [epsilon, delta]
    if any(nnz(weights) < epsilon) || any(weights > delta)
        fitness = fitness + 1000 * (sum(nnz(weights) < epsilon) + sum(weights > delta));
    end

    % Check the returns constraints
    if returns > max_returns
         fitness = fitness + 10000 * (1+returns/max_returns);
    end

    % Penalty for variance outside bounds
    if variance > max_variance
        fitness = fitness + 10000 * (1+variance/max_variance);
    end
end
```

Efficient Frontier calculation

- **Division of the Efficient Frontier:** obtain 100 returns objective in the min-max range returns
- **Solve the optimization problem:** obtain the minimum variance portfolio for each level of return

Efficient Frontier and Implemented PSO



Critique and Limitations

- **Financial considerations:**
  - *Assumption of Market Data:* Relies on historical returns and covariance which may not be accurate predictors of future performance.
  - *Data quality:* Only one brief time span (5 years), no consideration of survivor bias and weekly price data.
- **Methodological considerations:**
  - *Parameter Sensitivity:* Heavily reliant on parameter settings (e.g., w, $c1$, $c2$).
  - *Scalability:* Performance degrade with larger portfolio sizes.
  - *Fitness function:* No clear presentation of this crucial setting